Table 2: Timing Results for 6 of 17 experimental studies

Experiment #	1	3	4	5	8	16
NIV	3	3	3	3	2	3
DD time (secs)	4056	2948	4312	5300	24610	5604
DD iters	1000	1000	850	1000	650	850
t_{DD}	.0000984	.0001052	.0001231	.0001286	.0001213	.0001212
AD time (secs)	10376	9994	10156	9700	35430	10130
AD iters	575	710	425	425	650	850
t_{AD}	.0004377	.0005021	.0005797	.0005536	.0001747	.0002190
relative work ratio ($t_{AD}/t_{DD)}$	4.449	4.775	4.710	4.306	1.440	1.807

Table 3: DD and AD timings for geometric sensitivities with Baldwin-Lomax turbulence model on 97x25x17 grid, NIV ranges from 1 to 48. Flow solver was executed for 50 iterations.

NIV	1	2	4	6	8	12	48
DD time	174	261	435	609	783	1131	4263
AD time	254	360	593	2034	2230	2298	3846
relative work ratio (t_{AD}/t_{DD})	1.46	1.38	1.36	3.34	2.85	2.03	.902

Table 1: Numerical results. (17 Experimental Studies)

	exp #	flow case	flow type	grid size	C_L			C_D			C_M		
Non-geometric Sensitivities					M	α	Re	M	α	Re	M	α	Re
	1	1	I	97x25x17	1.0003	.9999		1.0000	.9999		1.0002	.9999	
	2	2	L	97x17x17	*	1.0000	*	1.0004	*	1.0000	*	1.0000	*
	3	3	L	97x17x17	1.0028	1.0000	1.0000	.9998	1.0001	1.0000	.9985	1.0000	1.0001
	4	4	T,ML	97x25x17	1.0000	1.0000	1.0007	1.0000	1.0000	1.0000	.9999	1.0000	1.0012
	5	4	T,BL	97x25x17	1.0000	1.0000	.9991	1.0000	1.0000	1.0000	1.0000	1.0000	.9961
Algorithmic Sensitivities		flow case	flow type	grid size	C_L			C_D			C_M		
					VIS2		VIS4	VIS2		VIS4	VIS2		VIS4
	6	4	T,BL	97x25x17	.9972		.9943	1.0008		1.0015	.9954		1.0016
Turbulence Modeling Parameter Sensitivities		flow case	flow type	grid size	C_L			C_D			C_M		
					K		A+	K		A+	K		A+
	7	4	T,BL	97x25x17	.9980		.9997	.9882		.9950	.9879		.9997
	8	4	T,BL	193x49x33	.9999		.9997	1.0000		1.0001	.9998		.9999
	9	5	T,BL	193x49x33	.9997		.9996	.9998		1.0027	.9997		.9996
	10	4	T,JK	193x49x33	.7874		1.0083	.7750		1.0403	.8037		.7359
	11	4	T,JK	193x49x33	.9940		.9694	.9826		.9544	.9935		1.0191
	12	5	T,JK	193x49x33	1.0513		.7680	.9512		-.0807	1.0355		.7363
Geometric Sensitivities (planform)		flow case	flow type	grid size	C_L			C_D			C_M		
					c_{tip}	$x_{tip,LE}$	$z_{tip,LE}$	c_{tip}	$x_{tip,LE}$	ztip,LE	c_{tip}	$x_{tip,LE}$	$z_{tip,LE}$
	13	1	I	97x33x17	1.0000	1.0000	1.0000	1.0074	1.0000	1.0000	1.0000	1.0000	1.0000
	14	4	T,BL	97x33x17	.9939	1.0062	.9745	.5938	1.0007	.7690	.9629	.9981	1.0781
	15	6	T,BL	97x33x17	1.0091	1.0033	1.0005	.7066	1.0005	.8238	1.0105	.9988	1.0170
Geometric Sensitivities (section)		flow case	flow type	grid size	C_L			C_D			C_M		
					α_{root}		α_{tip}	α_{root}		α_{tip}	α_{root}		$\alpha_{\tau ip}$
	16	4	T,BL	97x33x17	.9996		1.0009	.9996		1.0001	.9991		1.0012
	17	6	T,BL	97x33x17	.9998		1.0008	.9997		1.0002	.9995		1.0011

1. ONERA M6, M = .84, α = 3.06°, inviscid
2. ONERA M6, M = .20, α = 0°, Re = 5000
3. ONERA M6, M = .20, α = 1°, Re = 5000
4. ONERA M6, M = .84, α = 3.06°, Re = 11.7x10^6
5. ONERA M6, M = .84, α = 5.06°, Re = 11.7x10^6
6. ONERA M6, NACA 00098 Airfoil, M = .84, α = 3.06°, Re = 11.7x10^6

(As described in Subsection 5.3, each number in the shaded portion of the table is a ratio of the AD–generated derivative value D_{AD} to the DD–generated derivative value D_{DD}. In experiment 1, the blank entries indicate derivatives that were computed to be zero by both D_{AD} and D_{DD}. In experiment 2, each "*" indicates a ratio having a highly suspect D_{DD} value.)

Application," SIAM, Philadelphia, PA, 1991.
18. D. Juedes, "A Taxonomy of Automatic Differentiation Tools," In A. Griewank and G.F. Corliss, eds., *Proceedings of the Workshop on Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, SIAM, Philadelphia, PA, pp. 315–329, 1991.
19. A. Carle, K.D. Cooper, R.T. Hood, K. Kennedy, L. Torczon and S.K. Warren, "A Practical Environment for Scientific Programming," *IEEE Computer*, 20(11), pp. 75–89, Nov. 1987.
20. D. Callahan, K. Cooper, R.T. Hood, K. Kennedy and L.M. Torczon, "ParaScope: A Parallel Programming Environment," *International Journal of Supercomputer Applications*, 2(4), Dec. 1988.
21. T.F. Coleman and J.J. Moré, "Estimation of Sparse Jacobian Matrices and Graph Coloring Problems," *SIAM Journal on Numerical Analysis*, 20:187–209, 1984.
22. T.F. Coleman, B.S. Garbow and J.J. Moré, "Software for Estimating Sparse Jacobian Matrices," *ACM Transactions on Mathematical Software*, 10:329–345, 1984.
23. B. Averick, J. Moré, C. Bischof, A. Carle and A. Griewank, "Computing Large Sparse Jacobian Matrices using Automatic Differentiation," ANL–MCS–P348–0193, Mathematics and Computer Science Division, Argonne National Laboratory, 1993. Also to appear in *SIAM Journal of Scientific Computing*.
24. V.N. Vatsa and B.W. Wedan, "Development of a Multigrid Code for 3–D Navier-Stokes Equations and Its Applications to a Grid-Refinement Study," *Computers & Fluids*, 18(4), pp. 391–403, 1990.
25. B. Baldwin and H. Lomax, "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flow," AIAA 78–257, 1978.
26. D. Johnson and L. King, "A Mathematically Simple Turbulence Closure Model for Attached and Separated Turbulent Boundary Layers," *AIAA Journal*, vol. 23, no. 11, pp. 1684–1692, 1985.
27. V.N. Vatsa, M.D. Sanetrik and E.B. Parlette, "Development of a Flexible and Efficient Multigrid-Based Multiblock Flow Solver," AIAA 93–0677, Jan. 1993.
28. J-Ch. Gilbert, "Automatic Differentiation and Iterative Processes," *Optimization Methods and Software*, Vol. 1, pp. 13–22, 1992.
29. B.D. Christianson, "Reverse Accumulation and Accurate Rounding Error Estimates for Taylor Series Coefficients," *Optimization Methods and Software*, 1(1):81–94, 1992.
30. A. Griewank, C. Bischof, G. Corliss, A. Carle and K. Williamson, "Derivative Convergence for Iterative Equation Solvers," *Optimization Methods and Software*, Vol. 2, pp. 321–355, 1993.
31. B. Averick, R.G. Carter and J.J. Moré, "The MINPACK-2 Test Problem Collection (Preliminary Version)." Technical Report ANL/MCS–TM–150, Mathematics and Computer Science Division, Argonne National Laboratory, 1991.

8. Acknowledgments

We would like to thank John Dennis and Ken Kennedy of Rice University, and Jorge Moré of Argonne National Laboratory for supporting the implementation of ADIFOR; George Corliss and Andreas Griewank for their seminal roles in the development of ADIFOR; Veer Vatsa of CAB/FldMD at NASA Langley for numerous useful and informative discussions concerning the TLNS3D and TLNS3D-MB codes; and Kitty Haigler of CSB/FldMD at NASA Langley for her continued support of the AD effort. We further thank several people of AAOB/SDyD including Dr. J.-F. Barthelemy, Laura Hall, and Eric Unger (now with McDonnell Douglas) for their pioneering efforts with AD at NASA Langley.

The work of C. Bischof was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, by the National Aerospace Agency under Purchase Order L25935D.

The work of A. Carle was supported by the National Aerospace Agency under Cooperative Agreement No. NCCW-0027, and by the National Science Foundation under cooperative agreement CCR-9120008.

9. References

1. J.S-. Sobieski, "Multidisciplinary Optimization for Engineering Systems: Achievements and Potential," NASA TM 101566, NASA, Mar. 1989.
2. S. Dollyhigh, J.S-. Sobieski, "Recent Experience with Multidisciplinary Analysis and Optimization in Advanced Aircraft Design," In *Third Air Force/NASA Symposium on Recent Advances in Multidisciplinary Analysis and Optimization. A Collection of Technical Papers*, San Francisco, CA, pp. 404–411. Sept. 1990.
3. P.G. Coen, "Recent Results from the High-speed Airframe Integration Research Project," AIAA Paper 92–4717, Sept. 1992.
4. P.G. Coen, J.S-. Sobieski and S. Dollyhigh, "Preliminary Results From the High-Speed Airframe Integrated Research Project," AIAA 92–1004, Feb. 1992.
5. J. Sobieszczanski-Sobieski, J.-F. Barthelemy and K.M. Riley, "Sensitivity of Optimum Solutions to Problem Parameters," *AIAA Journal*, Vol. 20, No. 9, pp. 1291–1299, Sep. 1982.
6. J.-F. Barthelemy, J. Sobieszczanski-Sobieski, "Optimum Sensitivity Derivatives of Objective Functions in Nonlinear Programming," *AIAA Journal*, Vol. 21, No. 6, pp. 913–915, June 1983.
7. L. Green, C. Bischof, A. Carle, A. Griewank, K. Haigler and P. Newman, "Automatic Differentiation of Advanced CFD Codes With Respect to Wing Geometry Parameters for MDO," *Abstracts from Second U.S. National Congress on Computation Mechanics*, Washington, D.C., p. 136, August 16–18, 1993.
8. C. Bischof, G. Corliss, L. Green, A. Griewank, K. Haigler and P. Newman, "Automatic Differentiation of Advanced CFD Codes for Multidisciplinary Design," *Computing Systems in Engineering* 3(6), 1993. Also presented at the Symposium on High-Performance Computing for Flight Vehicles, Arlington, VA, Dec. 7–9, 1992.
9. L. Green, P. Newman and K. Haigler, "Sensitivity Derivatives for Advanced CFD Algorithm and Viscous Modeling Parameters via Automatic Differentiation," AIAA 93–3321, 1993.
10. V.M. Korivi, A.C. Taylor III, P.A. Newman, G.J.-W. Hou and H.E. Jones, "An Approximately-Factored Incremental Strategy for Calculating Consistent Discrete Aerodynamic Sensitivity Derivatives," In *Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, Cleveland, OH, AIAA 92–4746–CP, pp. 465–478, Sept. 1992.
11. P.A. Newman, G.J.-W. Hou, H.E. Jones, A.C. Taylor III- and V.M. Korivi, "Observations on Computational Methodologies For Use In Large-Scale Gradient-Based Multidisciplinary Design," In *Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, Cleveland, OH, AIAA 92–4753–CP, pp. 531–542, Sept. 1992.
12. L.B. Rall, "Automatic Differentiation: Techniques and Applications," *Volume 120 of Lecture Notes in Computer Science*, Springer Verlag, Berlin, Germany, 1981.
13. C.H. Bischof, A. Carle, G.F. Corliss, A. Griewank and P. Hovland, "ADIFOR: Generating Derivative Codes from Fortran Programs," *Scientific Programming*, 1(1), pp. 1–29, 1992.
14. C.H. Bischof, A. Carle, G.F. Corliss and A. Griewank, "ADIFOR: Automatic Differentiation in a Source Translator Environment," In P. Wang, ed., *International Symposium on Symbolic and Algebraic Computing 92*, ACM, Washington, D.C., pp. 294–302, 1992.
15. C.H. Bischof and A. Griewank, "ADIFOR: A Fortran System for Portable Automatic Differentiation," in *Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Optimization*, Cleveland, Ohio, AIAA 92–4744–CP, pp. 433–441, Sept. 1992.
16. A. Griewank, "On Automatic Differentiation," In M. Iri and K. Tanabe, eds., *Mathematical Programming: Recent Developments and Applications*, Kluwer Academic Publishers, Boston, MA, pp. 83–108, 1989.
17. A. Griewank and G.F. Corliss, eds., "Automatic Differentiation of Algorithms: Theory, Implementation, and

with respect to non-geometric inputs, geometric inputs, and algorithmic and physics modeling parameters. Our experiments demonstrate that the AD method has several distinct advantages in comparison to DD:

- AD requires a short, initial, code-development lead time to obtain SD codes capable of providing accurate SD. Our initial application of ADIFOR to TLNS3D required *O*(man-month) time. Subsequent efforts required *O*(man-week) or less.
- Accurate SD can be computed without the need to determine a step-size for the DD perturbation capable of providing an accurate derivative approximation. This is especially important in cases, such as the experiment to compute geometric sensitivities with respect to planform derivatives, in which DD failed to produce accurate approximations to the SD's despite several attempts with different perturbation sizes.
- Accurate SD can be computed for code that has been deemed to be too complicated to be differentiated by hand. For example, in the 2–D quasi–analytical SD code of reference 10, the turbulence model was deemed too complicated for differentiation by hand; its treatment as constant led to sizable relative errors in some resulting global sensitivities. ADIFOR, however, successfully differentiated the TLNS3D Baldwin-Lomax turbulence model providing answers consistent with those produced by DD.
- Small derivative values can easily be calculated. DD accuracy for small derivatives is often suspect.
- "Consistent" derivatives of oscillatory output functions can be obtained. For cases for which the convergence was poor due to a noisy output, the DD based upon instantaneous output values failed to adequately approximate the SD. The AD code appeared to capture several meaningful digits of each SD computed for the turbulent case using the noisy Johnson–King turbulence model.
- Convergence of the SD via AD can be monitored during code execution, whereas SD via DD are usually constructed after the runs are complete, and this precludes convergence monitoring during execution.

Unfortunately, at this stage, the time required for an AD-augmented code to compute derivatives is still larger than we would like it to be for codes as complex as TLNS3D. As reported above, for a sufficiently large number of design variables, AD outperforms DD; however, the memory required for large numbers of design variables, is prohibitive. AD fares far better if all of the costs actually incurred by using DD are considered:

- Time and space required to manage the data from multiple perturbed function evaluations.
- Time required to verify that the DD step-size is providing accurate SD approximations.
- Time required to compute DD by central differences, if one-sided divided differences are not able to provide accurate SD approximations.

It should be stressed that, from a purely mathematical point of view, the differentiation of iterative processes does not seem to be a problem, despite the fact that the assumptions of known derivative convergence theorems have not been verified and are almost certainly not satisfied by multigrid algorithms. Since even the convergence of the iterates themselves has not been proven under reasonably general assumptions, attempts to prove the convergence of their derivatives seem premature. As our theoretical studies and numerical experiments indicate, one may expect that both solutions and derivatives converge at about the same rate once the iteration has settled down.

7. Future Work

We are considering several approaches to reduce the cost of AD for iterative flow solvers.

First, we are investigating the "by-hand" application of AD in the context of an incremental iterative strategy. This approach should have two beneficial effects. Since less code will be processed by AD, the space required will be significantly reduced. This reduction in space, in turn, means that the number of design variables can be increased, leading to a significant improvement in vector efficiency and consequent reduction in computational effort.

Second, we are investigating automatic "deactivation" concepts in which an attempt will be made to avoid the unnecessary differentiation of preconditioners and other intermediates that affect only the solution process but not the solution function and its derivatives. Unless the original code is appropriately structured, "deactivating" such intermediates will be a difficult task. However, the resulting simplified derivative calculation should be very efficient.

Third, we are investigating techniques to improve the vectorization and parallelization of the derivative code, so that their running time is at worst equal to that of the original code multiplied by the number of design parameters.

points. Similarly, define T_{AD} to be the time required to compute sensitivities by AD taking *I* multigrid iterations over a grid with G grid points. Then, define t_{DD} and t_{AD} as follows.

$$t_{DD} = \frac{T_{DD}}{(G \times I)} \tag{5}$$

and

$$t_{AD} = \frac{T_{AD}}{(G \times I)} . \tag{6}$$

The ratio of t_{AD} to t_{DD}, referred to as the "relative work ratio," indicates the relative amount of work performed by an iteration of the AD code at a grid point to that performed by an iteration of the DD code at a grid point.

Table 2 presents timing results from five of the experimental sensitivity studies. Again, each experiment is identified by experiment number. The inefficiency of the AD code, even with loop unrolling by the Cray compiler, is captured by the relative work ratios of 4.449, 4.775, 4.710 and 4.306 for non–geometric sensitivity experiments 1, 3, 4 and 5.

The Cray compiler Flowtrace and Loopmark options were used to identify subroutines, function calls, and "do loops" that did not vectorize as well as the corresponding ones in the original code and, thus, probably consumed far too much execution time in the AD code. After this evaluation, it was possible to use simple code modifications (changing one recurrent subroutine argument to a parameter, restoring intrinsic Cray vector functions which ADIFOR could not process) and more compiler options (use of the aggressive compile option, and inlining of the Fortran intrinsic and error handling functions provided by ADIFOR) to improve the derivative code vectorization. After this processing, some degradation of the vector inefficiency still exists, most likely caused by the inability of the Cray compiler to vectorize the more complex loops generated by ADIFOR. Manual loop segmentation, known as loop distribution in the compiler community, allows the compiler to vectorize more of the remaining loops. Additional segmentation may recover even more of the vector performance. Both the turbulence modeling parameter sensitivity studies and geometric sensitivity studies were executed with this "hand–tuned" code. The improved efficiency of this AD code is demonstrated by the relative work ratio of 1.440 for the viscous modeling parameter sensitivity of experiment 8 and the relative work load of 1.807 for the geometric sensitivity of experiment 16.

Experience gained through manually postprocessing the TLNS3D code will be incorporated into future versions of ADIFOR, thereby improving vector efficiency and decreasing user intervention.

To document the impact of number of independent variables on the efficiency of $\text{TLNS3D}_{\text{AD}}$, a series of experiments were run with number of independent variables ranging from 1 to 48 as shown in Table 3. It is significant to note how the time per design variable increases sharply in going from 4 to 6 design variables, and then slowly decreases as the number of design variables increases beyond six. This is due to the Cray compiler which will only automatically unroll the innermost loops up to length 5. However, as the time per design variable indicates, extremely good vectorization was achieved with the 48 design variable case.

Memory requirements for the AD codes created in the experimental studies, in general, required between NIV and NIV+1 times as much memory as that required for the original TLNS3D code. One would expect that the memory would increase by a factor of about NIV+1 over the original code in order to accommodate calculation and storage of the function plus the NIV derivatives by an iterative scheme. ADIFOR dependence analysis reduces the memory requirements somewhat by augmenting only the "relevant" portion of the function code.

5.5 Ease–Of–Use

The experimental sensitivity studies demonstrated that reliable (and verified) SD could be obtained for complex CFD codes in *O*(man–week), an enormous improvement over the time required to construct a "by-hand" derivative code. After an initial successful application of ADIFOR to a code such as TLNS3D, subsequent applications to compute new sets of sensitivities should take only *O*(man–hours) which is quite competitive with divided differences, but is free of the uncertainty inherent in the use of divided differences.

As an indication of the ease–of–use of ADIFOR and the unreliability of DD, the majority of time dedicated to performing each of these experiments was spent verifying the ADIFOR–generated derivatives by DD — not applying ADIFOR to generate the AD code.

6. Conclusions

Automatic differentiation of TLNS3D, an efficient, complex, state of the art 3-D CFD code, has been quantitatively demonstrated to provide accurate SD of output flow properties

ratio of the AD–generated derivative value D_{AD} to the DD–generated derivative value D_{DD}. A ratio near unity indicates good agreement between the AD and DD derivatives. Note that in almost all cases AD and DD were in agreement to between 3 and 4 significant digits of accuracy.

Experiments 1 through 14 were run using a restart-based approach in which the original TLNS3D code was run to reasonably good convergence and then a restart solution file was generated. Then, the SD augmented TLNS3D code was run from the restart file to converge the SD. Experiments 15 through 17 were run from a scratch start. For more information on these two approaches, again refer to references 7, 8 and 9.

In experiment 1, the blank entries indicate derivatives that were computed to be zero by both D_{AD} and D_{DD}. In experiment 2, each "*" indicates a ratio having a highly suspect D_{DD} value computed by taking function differences of magnitude $O(10^{-14})$. The sensitivities computed by AD have magnitude $O(10^{-8})$ and we believe them to be correct.

Experiments 10, 11 and 12 highlight the differences between the asymptotic convergence behavior of the Baldwin–Lomax and Johnson–King turbulence models. At the low angle–of–attack ($\alpha = 3.06°$) condition of experiments 10 and 11, the Johnson–King residual converges several orders of magnitude and then "hangs up" in an oscillatory low amplitude noise; the Baldwin–Lomax residual can be driven to machine zero without difficulty. At the high angle–of–attack ($\alpha = 5.06°$) condition of experiment 12, the amplitude of the noise is increased. Noise also appears in the computations for the dependent variables C_L, C_D and C_M. DD based on such a noisy function cannot be expected to give meaningful SD as indicated by the poor agreement of four of the six sensitivity ratios computed in experiment 10. However, the AD's are expected to give meaningful results because the SD calculations via AD are driven by the most significant digits of the function evaluation, whereas the DD are constructed by subtraction, where the most significant digits of the function evaluation are lost. In experiment 11, an attempt was made to factor out the oscillatory noise by "time–averaging" the dependent variables used in the DD calculation. Underlined values in Table 1 indicate ratios calculated using D_{DD} values computed using time–averaged function values computed by a sequence of function iterations. Time–averaging significantly improved the agreement between AD and DD in this case. Unfortunately, time–averaging had little affect on the agreement between AD and DD in experiment 12.

Experiments 14 and 15 demonstrate a case where DD may be difficult to use to obtain accurate SD approximations due to the use of a highly stretched grid. The DD used in these ratios were one-sided forward differences, but recent calculations using one-sided backward, and central differences show little improvement over these results. We believe the sensitivities of the wing drag coefficient C_D to the tip leading edge spanwise location $z_{tip,LE}$ computed by AD to be correct.

The numerical results reported here show that even the naive application of ADIFOR to multigrid solvers viewed essentially as black–box programs can produce accurate sensitivity information at tolerable costs. In fact, for the majority of the sensitivity studies, all calculated derivatives could be reproduced with several digits agreement by carefully evaluated DD.

5.4 Running Time and Storage Requirements

To generate a code that computes sensitivities with respect to a set of independent variables of size NIV, ADIFOR inserts vector loops of length NIV to compute "gradient objects" for each intermediate involved in the function evaluation, using roughly NIV times as much memory as the original code. If NIV is sufficiently large, then the resulting AD code will be an efficient vector code. For standard test problems, ADIFOR achieves and undercuts this bound regularly on scalar and super–scalar chips.[31] Unfortunately, the large memory requirements of TLNS3D and the factor of NIV expansion in memory required for the forward mode of AD, practically limits the number of independent variables that can be computed by $TLNS3D_{AD}$ to about 10 for reasonably fine grids. Therefore, all of the vector loops inserted by ADIFOR are "short loops." With current Cray compiler technology, these inner–most short loops prevent the longer TLNS3D outer loops from being pipelined leading to an inordinately long running time. For up to 5 independent variables, significant improvements can be made to the AD code by making minor code changes "by–hand" and by directing the Cray compiler to unroll the short vector loops. Unfortunately, these changes are insufficient to create an efficient vector version of $TLNS3D_{AD}$.

Since the multigrid solver and its derivatives appear to converge at roughly the same asymptotic rate for smoothly converging functions, the cost of AD versus DD can be assessed by comparing the cost of a "single iteration" of the AD code and the DD code normalized based on grid size. The concept of "single iteration" of the AD code is well defined. For the DD code, the cost of a "single iteration" is the sum of the costs of each iteration of the NIV+1 perturbed function evaluations required to compute the one-sided DD approximations.

Define T_{DD} to be the time required to compute sensitivities by DD taking I multigrid iterations over a grid with G grid

investigated derivatives with respect to free stream Mach number *M*, angle of attack α, and stream Reynold's number based on mean aerodynamic chord *Re*. Derivatives were verified for inviscid transonic flow, laminar subsonic flow and turbulent transonic flow. Both the differentiable mixing–length turbulence model and the Baldwin–Lomax turbulent models were compared for the transonic flow case.

The algorithmic parameter sensitivity study investigated the second– and fourth–order damping coefficients (VIS2 and VIS4) of the CFD solution algorithm of the TLNS3D code which uses a blending of scalar second– and fourth–difference artificial dissipation to maintain numerical stability. Derivatives were verified for transonic turbulent flow.

The physics modeling parameter sensitivity study investigated two turbulence parameters: the Clauser constant K of the outer–region eddy viscosity coefficient and A+ of the Van Driest correction to the Prandtl mixing length of the inner–region viscosity coefficient. Derivatives were verified for transonic turbulent flow. The Baldwin–Lomax turbulence model was examined on both the 97x25x17 and 193x49x33 grids. The Johnson–King turbulence model was only investigated on the 193x49x33 grid due to the excessive noise encountered while trying to converge the residual on the coarser grid.

The application of ADIFOR for geometric inputs investigated derivatives with respect to planform and section parameters. The planform parameters are the tip chord, the tip leading edge streamwise location, and the tip leading edge spanwise location (c_{tip}, $x_{tip,\,LE}$, and $z_{tip,\,LE}$). The section parameters are the twist angle, maximum camber, location of maximum camber, and the thickness-to-chord ratio at the wing root and tip (α_{root}, $cmax_{root}$, $xcmax_{root}$, τ_{root}, α_{tip}, $cmax_{tip}$, $xcmax_{tip}$, τ_{tip}). Derivatives with respect to planform were computed for both the original ONERA M6 wing and an ONERA M6 planform with a NACA 00098 wing section (that is, a symmetric "four–digit" airfoil of thickness–to–chord ratio 0.098) that approximates the original ONERA M6 wing section. Both inviscid and turbulent transonic flows were studied.

All ADIFOR preprocessing of codes for these experiments was performed on a SPARC workstation. All TLNS3D results were obtained on one processor of the NASA Langley Research Center Cray Y–MP.

5.2 ADIFOR Code Processing

In preparation for the non–geometric sensitivity studies, ADIFOR was applied to TLNS3D in a very simple and straightforward manner, with the non–geometric inputs identified as the independent variables, to construct the sensitivity code $TLNS3D_{AD}$. Prior to processing with ADIFOR, minor changes to the TLNS3D code required for ADIFOR were performed. ADIFOR differentiated through the entire multigrid solution algorithm; the specified dependences were traced from independent to dependent variables and the SD code inserted as required. The resulting SD modules were assembled into a working code based on the template shown in Figure 3. The SD code was later modified to improve its performance on the Cray Y–MP as described later in this section.

For the algorithmic and physics modeling parameter sensitivity studies, ADIFOR was again applied to TLNS3D, with the algorithmic and physics modeling parameters identified as independent variables, to construct the sensitivity code $TLNS3D_{AD}$.

In preparation for ADIFOR processing for the geometric sensitivity study, ADIFOR was applied to WTCO to generate the new code $WTCO_{AD}$ with the geometric parameters specified as the independent variables, and the grid coordinates (x, y, z) generated by WTCO specified as the dependent variables. In addition to generating a grid, $WTCO_{AD}$ also generates the derivatives or grid sensitivities to the geometric parameters (either planform or section). ADIFOR is then applied to TLNS3D, to create $TLNS3D_{AD}$, with the grid coordinates declared as the independent variables and the flow coefficients as the dependent ones. Both grid and grid sensitivities are read as input by $TLNS3D_{AD}$. The grid sensitivities initialize the seed matrix within $TLNS3D_{AD}$ so that an application of the chain rule can be used to calculate the required SD's as

$$\frac{dC_L}{dG} = \frac{dC_L}{dX} \cdot \frac{dX}{dG} \tag{4}$$

where G is a generic geometric variable and X is the grid generated by $WTCO_{AD}$ and used as the input to $TLNS3D_{AD}$. The grid sensitivity array $\frac{dX}{dG}$ is the seed matrix input to $TLNS3D_{AD}$.

5.3 Accuracy

Table 1 presents a summary of seventeen experiments investigating the ADIFOR–generated codes $WTCO_{AD}$ and $TLNS3D_{AD}$. Each experiment has been assigned a number given in the "exp #" column. The flow case column identifies the configuration and flow condition as indicated in the legend under the table. Flow type labels I, L, T_{ML}, T_{BL} and T_{JK} indicate inviscid transonic flow, laminar subsonic flow, and turbulent transonic flow modeled with a mixing–length model, Baldwin–Lomax model, or Johnson–King model, respectively. Each number in the shaded portion of the table is a

For the sake of discussion, assume that our iteration for solving Equation 1 and throughout has the schematic form shown in Figure 2.

```
for m = 1, ... do
    evaluate R(z_m, x_*) and stop if it is small
    compute a suitable preconditioner P_m
    update z_{m+1} = z_m − P_m R(z_m, x_*)
endfor
```

FIGURE 2. Original iteration

Let the notation R_z and R_x represent $\frac{\partial R}{\partial z}$ and $\frac{\partial R}{\partial x}$, respectively. Newton's method, for example, is a particular instance of this scheme with P_m defined as follows.

$$P_m = \left(\left. \frac{\partial R}{\partial z} \right|_{z = z_m} \right)^{-1} \tag{3}$$

In the following, a "prime" notation (such as z') always denotes total differentiation with respect to x. Applying AD to the schematic iteration shown above, and modifying the stopping criterion, provides the derivative iteration shown in Figure 3.

```
for m = 1, ... do
    R_m = R(z_m, x_*)
    R'_m = R_z(z_m, x_*) z'_m + R_x(z_m, x_*)
    if R_m and R'_m are small enough stop
    compute P_m and its derivative P'_m
    z_{m+1} = z_m − P_m R_m
    z'_{m+1} = z'_m − P'_m R_m − P_m R'_m
endfor
```

FIGURE 3. SD augmented iteration

Given z_m and z'_m, one can obtain the derivative residual R'_m at a cost roughly equal to that of evaluating R multiplied by the number of design parameters (i.e., components in x). In particular, this derivative evaluation does not require the calculation of the Jacobian R_z, which may contain very many elements. The stopping criterion based solely on R_m has been replaced by one that also requires R'_m to be small. While it is natural to do so, an automatic tool cannot be expected to detect the stopping criterion in a potentially complicated code without some user intervention. Conceptually, one may remove the stopping criterion completely to obtain infinite sequences of iterates z_m and derivative approximations z'_m, which have been shown to converge R–linearly.[30] This result was originally obtained by Gilbert[28] and Christianson[29] for the case of Newton's method and similar smooth fixed point iterations.

These results have been extended to quasi–Newton methods, where the derivatives P'_m may grow unbounded but $P'_m R_m$ still tends to zero, because of the superlinear rate of convergence.[30] Whenever the iterates themselves converge superlinearly there is the danger that the R–linearly convergent derivative approximations may lag behind. For such methods, it is particularly important that the stopping criterion enforce a significant reduction of $\|R'_m\|$. In large–scale applications, a reasonable linear rate is often the best one can achieve, so that the asymptotic rate of convergence is likely to be the same.

5. Experimental Studies

In order to evaluate the promised abilities of AD and ADIFOR to deliver SD of complex numerical processes accurately and efficiently, a sequence of experiments was performed in which ADIFOR was applied to the WTCO wing grid generation code and TLNS3D to compute the sets of derivatives described above. Highlights from those experiments will now be presented. For more complete information on the results of these studies, refer to references 7, 8 and 9. For example, in this overview, almost all of the investigations into the question of what actually defines "good convergence for the flow solver" and "good convergence for the ADIFOR–generated codes" are ignored.

The primary concerns about the use of AD in computing SD were accuracy, running time requirements and memory requirements of the derivative code generated by ADIFOR. Ease–of–use was a secondary concern. Accuracy and running time were assessed by comparing the results of the ADIFOR–generated code with DD.

5.1 The Experiments

Each of the experimental studies investigated derivatives given an ONERA M6 wing described by a C–O mesh. The lift coefficient C_L, drag coefficient C_D and wing pitching moment C_M were taken as the output of TLNS3D with respect to differentiation.

The application of ADIFOR for non–geometric sensitivities

properties such as improved convergence rate or improved accuracy.

In the studies described below, the WTCO wing grid generation code has been coupled with the TLNS3D thin–layer Navier–Stokes code.

The WTCO wing grid generation is a batch–mode, algebraic, transfinite interpolation grid generation program. The code includes the capability to generate grids around wings of at least two airfoil sections, described by pairs of coordinates or NACA four–digit airfoils. In this case, the usual NACA four–digit airfoil family has been expanded by allowing the maximum camber (cmax), location of maximum camber (xcmax), and the thickness–to–chord ratio (τ) to be specified as real number inputs, rather than deduced from an integer designation. A true NACA 2412 has the cmax = 0.02, xcmax = 0.4, and $\tau = 0.12$; the WTCO program has been modified to allow for small perturbations to this shape, such as cmax =0.02002, xcmax = 0.4004, and $\tau = 0.12012$. These modifications were necessary to perform the DD runs used to verify the SD calculations by AD. The user specifies spacing constants at several points on the grid boundaries as well as the type of boundary. The user also specifies the type of boundary surface, interior interpolation, and the amount of smoothing required. WTCO was chosen for use in this study because it has performed reliably with the flow solver on many previous occasions; it is non–iterative and does not include any coding extraneous to the grid generation (such as graphics). The source code necessary for AD application was readily available. The total number of design variables can be changed by changing the number of input sections which describe the wing.

The TLNS3D code is a high–fidelity aerodynamic computer program that solves the time–dependent 3–D thin–layer Navier–Stokes equations with a finite–volume formulation.[24] The code employs grid sequencing, multigrid, and local time stepping to accelerate convergence and efficiently obtain steady–state high Reynolds number turbulent flow solutions. When temporally converged to a steady–state solution, the method is globally second–order accurate. The TLNS3D code is a central–difference code that employs second–order central differences for all spatial derivatives and employs a blending of scalar second– and fourth–difference artificial dissipation to maintain numerical stability. The solution is advanced explicitly in time with a five–stage Runge–Kutta time–marching algorithm. The code includes both the Baldwin–Lomax (B–L) and Johnson–King (J–K) turbulence model.[25,26] This code has been used successfully in a number of applications across the flight speed range from low subsonic to hypersonic and for a number of flight vehicle types. The TLNS3D code is a highly vectorized code and this aspect contributes greatly to its overall computational efficiency on the Cray Y–MP. The newly released multiblock version of TLNS3D, TLNS3D-MB[27], promises the flexibility needed for modeling complex geometric configurations and is basically organized in incremental iterative form, which may allow for significant improvements in the ADIFOR applications in the future.

4. Differentiation of Iterative Functions

Despite the lack of convergence theory under realistic assumptions, iterative techniques are now the state of the art in 3–D CFD codes for nontrivial geometries and stream conditions.[24] The iterative flow solvers may take hundreds of steps and often involve discontinuous adjustments of solution operators, grids, shock waves, or free boundaries. This iterative nature of advanced CFD codes poses the most significant challenge regarding the automatic generation of sensitivities.

If applied to a CFD code as a “black-box,” ADIFOR would simply differentiate the whole iterative process. Unfortunately, the resulting sensitivity code would only apply the stopping criterion from the original iterative process which has been shown to be insufficient to guarantee convergence of the sensitivities themselves. If the stopping criterion of the sensitivity code is modified to monitor convergence of the sensitivities as well, then the sensitivities computed by AD should be accurate.[28,29] Theoretical results on the rate of convergence of derivatives for iterative processes have not yet been extended to multigrid methods, such as that underlying TLNS3D, but our experience so far indicates that for smoothly converging functions, the derivatives and the function converge at roughly the same asymptotic rate.[9]

Consider a function of x_* defined implicitly by the following nonlinear system.

$$R(z, x_*) = 0 \tag{1}$$

The goal of an iterative solver is to find the value $z_* = z(x_*)$ of the function implicitly defined by R. The question is under what circumstances does an AD version of the code for this rootfinding process compute the following desired derivatives.

$$z'_* = \left.\frac{dz}{dx}\right|_{x = x_*} \tag{2}$$

implementation is usually substantial.

In contrast to the approximation of derivatives by DD, AD does not incur any truncation error so that the resulting derivative values are usually obtained with the working accuracy of the original function evaluation. In contrast to fully symbolic differentiation, both operations count and storage requirements can be bounded a priori in terms of the complexity of the original function code for all modes of AD. In many cases, the calculations initiated by an AD tool for the evaluation of derivatives mirror those of a carefully handwritten derivative code. For more information, refer to the comprehensive collection on AD theory, implementation, and applications[17], and a review of the earlier AD tools[18].

ADIFOR uses sophisticated program analysis techniques to extract control flow and dependence information directly from the user's source code to determine which intermediate variables within a program require the propagation of derivative information.[19,20] This approach allows for a simple, intuitive interface and may greatly reduce the space and time requirements of the derivative code by eliminating the need to compute derivatives for all intermediate quantities in the program. Application of ADIFOR to Fortran codes requires specifying the independent and dependent variables to be used in forming the SD.

The ADIFOR tool produces portable Fortran 77 code and accepts almost all of Fortran 77, in particular, arbitrary calling sequences, nested subroutines, common blocks, and equivalences. The ADIFOR–generated code employs a consistent subroutine naming scheme that allows for code tuning, the use of domain–specific knowledge, and the exploitation of vendor–supplied libraries. ADIFOR–generated code can be used in various ways. Instead of simply producing code to compute the Jacobian *J*, ADIFOR produces code to compute *J * S*, where the "seed matrix" *S* is initialized by the user. Therefore, if *S* is the identity, ADIFOR computes the full Jacobian; whereas if *S* is just a vector, ADIFOR computes the product of the Jacobian by a vector. "Compressed" versions of sparse Jacobians can be computed by exploiting the same graph coloring techniques that are used for DD approximations of sparse Jacobians.[21,22] The running time and storage requirements of the ADIFOR–generated code are roughly proportional to the number of columns of *S*, so the computation of Jacobian–vector products and compressed Jacobians requires much less time and storage than does the generation of the full Jacobian matrix.[23]

Although AD techniques can be applied to augment codes in their entirety, significant reduction in the space and time required to compute sensitivities are often possible if domain–specific application knowledge is coupled with AD techniques in sophisticated ways. For instance, we expect that the derivative code created by applying AD to the components of a CFD code for use in an incremental iterative strategy will significantly outperform the code generated by fully differentiating the CFD iterative solver.

3. SD Requirements for CFD

The computational process of CFD, when coupled with a grid generator, maps geometric shape inputs, non–geometric stream inputs and a collection of algorithmic and physical modeling parameters to a flow solution as shown in Figure 1.

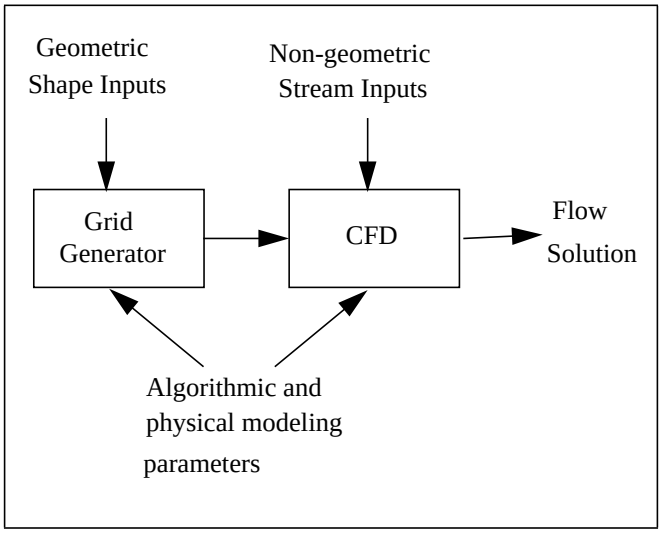

FIGURE 1. Block diagram of the CFD process

Geometric inputs include wing planform and wing section parameters. Non–geometric inputs include the free stream Mach number *M*, stream Reynold's number based on mean aerodynamic chord *Re*, and angle of attack α. Algorithmic and physical modeling parameters include the coefficients of the artificial dissipation terms and parameters used in turbulence modeling which control stability and convergence acceleration. The flow solution includes C_L, the wing lift coefficient, C_D, the wing drag coefficient, and C_M, the wing pitching moment. Interesting SD for the complete grid generation and CFD process include derivatives of any of the outputs (C_L, C_D or C_M) with respect to any of the geometric or non–geometric inputs or any of the algorithmic and modeling parameters. Geometric and non–geometric sensitivities might be used to guide an MDO system, for example, in determining an optimum wing shape. Sensitivities with respect to algorithmic and modeling parameters could be used in a formal parameter identification regime to achieve particular program

Numerous research efforts have examined the issue of efficient computation of SD for CFD, but most have concentrated on the use of direct solvers to solve the large systems of linear algebraic sensitivity equations generated by direct differentiation of the system of discrete nonlinear algebraic equations which model the Euler or thin–layer Navier–Stokes (TLNS) equations for 2–D flow. Unfortunately, the extremely large computer storage requirement of direct solvers make their extension to 3–D implausible. Advanced 3–D CFD codes, therefore, typically employ iterative solution algorithms to solve the implicit nonlinear partial differential equations that express the fluid conservation laws; it is exactly those iterative techniques that must be augmented with SD computations.

Typical techniques for "augmenting" a code to compute sensitivities include "by hand," "by use of a symbolic expression differentiator" and "by approximation via divided differences." Unfortunately, none of these techniques can be counted on to deliver fast and reliable derivatives in a flexible and timely fashion for large computer codes. Hand coding of derivatives is impractical and symbolic approaches may require as much effort as hand coding. Divided differences (DD) may not be accurate and are obtained too slowly. Since DD errors tend to grow with problem complexity, larger models will have to deal with ever–more–inaccurate derivatives, even though a faithful modeling of their complex nonlinear behavior requires very accurate derivatives. In addition, the cost of DD will restrain the magnitude of problems that can be done in practice.

Automatic differentiation (AD) promises to address the need for a flexible and scalable technology capable of computing derivatives of large codes accurately, irrespective of the complexity of the model. This paper presents an overview of a sequence of three efforts designed to determine the potential of AD with respect to the SD requirements of MDO for advanced CFD codes.[7–9] In each of these efforts, the ADIFOR automatic differentiation tool was applied to TLNS3D, a state of the art, 3–D, thin–layer Navier–Stokes, multigrid flow solver. In each of these efforts AD was applied to compute sensitivities of the entire iterative multigrid process. In contrast to this "black–box" approach, automatic differentiation could be applied to components of a CFD solver to obtain derivative computations in an incremental iterative form analogous to quasi-analytical hand differentiated code. The incremental iterative form, also known as the "delta" or "correction" form, is commonly used in CFD for obtaining the solution state vector from the nonlinear governing flow equations. References 10 and 11 discuss the benefits of using this form to solve the large systems of linear equations needed to obtain SD. Ongoing efforts are evaluating these alternative strategies.

This paper is organized as follows. Section 2 briefly overviews automatic differentiation and ADIFOR. Section 3 describes TLNS3D and examines sets of SD that might be needed in an MDO context. Section 4 discusses the application of AD to iterative CFD solvers. Section 5 presents results from the experimental studies in which ADIFOR was applied to compute sets of SD like those described in Section 3. Section 6 and Section 7 conclude the paper and present ideas for future work. As will be seen, the results of these efforts are both significant and encouraging; but challenges remain.

2. Automatic Differentiation and ADIFOR

Automatic differentiation[12] is a chain–rule–based technique for evaluating the derivatives of functions defined by computer programs with respect to their input variables and has been investigated since 1960. For the most part, implementations of AD were conceived by the need for accurate first– and higher–order derivatives in a certain application. Distribution for the mainstream of scientific computing was not a major concern. Recently, progress towards a general–purpose AD tool has been made with the development of ADIFOR by a joint effort of Argonne National Laboratory and Rice University.[13–15]ADIFOR differentiates programs written in Fortran 77; that is, given a Fortran procedure (or collection of procedures) that describe a "function" and an indication of which variables in parameter lists or common blocks correspond to "independent" and "dependent" variables with respect to differentiation, ADIFOR produces Fortran 77 code that computes the derivatives of the dependent variables with respect to the independent ones.

Automatic differentiation has two basic modes which are usually referred to as the forward and reverse modes. The forward mode computes derivatives of intermediate variables with respect to the independent variables during a "forward" pass through a function. In contrast, the reverse mode propagates derivatives of the final result with respect to intermediate quantities during a "reverse" pass through the function. Ignoring sparsity issues, the running time and storage requirements of the forward mode are roughly proportional to the number of independent variables. The reverse mode is closely related to adjoint methods and has a lower operations count for gradient computations, but potentially very large memory requirements.[16] ADIFOR employs a hybrid of the forward and reverse modes of AD. That is, for each assignment statement, code is generated for computing the partial derivatives of the result with respect to the variables on the right–hand side and then the partials are employed in the forward mode to propagate overall derivatives. The resulting decrease in complexity compared to an entirely forward mode

APPLICATIONS OF AUTOMATIC DIFFERENTIATION IN CFD

Alan Carle*
Rice University, Houston, Texas

Lawrence L. Green†
NASA Langley Research Center, Hampton, Virginia

Christian H. Bischof‡
Argonne National Laboratory, Argonne, Illinois

Perry A. Newman§
NASA Langley Research Center, Hampton, Virginia

Abstract

Automated multidisciplinary design of aircraft requires the optimization of complex performance objectives with respect to a number of design parameters and constraints. The effect of these independent design variables on the system performance criteria can be quantified in terms of sensitivity derivatives for the individual discipline simulation codes. Typical advanced CFD codes do not provide such derivatives as part of a flow solution. These derivatives are expensive to obtain by divided differences from perturbed solutions, and may be unreliable, particularly for noisy functions. In this paper, automatic differentiation has been investigated as a means of extending iterative CFD codes with sensitivity derivatives. In particular, the ADIFOR automatic differentiator has been applied to the 3–D, thin–layer Navier–Stokes, multigrid flow solver called TLNS3D coupled with the WTCO wing grid generator. Results of a sequence of efforts in which TLNS3D has been successfully augmented to compute a variety of sensitivities are presented. It is shown that sensitivity derivatives can be obtained accurately and efficiently using ADIFOR, although significant advances are necessary for the efficiency of ADIFOR–generated derivative code to become truly competitive with hand–differentiated code.

1. Introduction

In the past, design of flight vehicles typically required the interaction of many technical disciplines over an extended period of time in a more or less sequential manner. At present, computer–automated discipline analyses and interactions offer the possibility of significantly shortening the design cycle time, while simultaneous multidisciplinary design optimization (MDO) via formal sensitivity analysis (SA) holds the possibility of improved designs.

Procedures for MDO of engineering systems have been addressed by Sobieski and others.[1–6] Sobieski proposes a unified system SA guided by system sensitivity derivatives (SD); the optimizer code or algorithm that uses these SD is the outermost loop of the entire design process. The objective and constraint functions are now generally composed of output functions from several disciplines. Each single discipline analysis code is then to supply not only the output functions required for the constrained optimization process and other discipline analysis inputs, but also the derivatives of all of these output functions with respect to its input variables. These variables include not only the MDO variables, but also output functions from other disciplines that implicitly depend on the MDO variables. Thus, a key technology required for MDO procedures is the capability to calculate the SD of outputs from the various analysis codes with respect to a set of design variables. Since the envisioned flight vehicle concept determines which objectives, constraint functions, MDO design variables, and discipline analysis codes are required to model the pertinent physical aspects throughout the flight regime (i.e., the particular MDO problem), flexibility and automation, in addition to computational efficiency, are needed in SD computations.

---

* Research Scientist, Associate Member AIAA

† Research Scientist, Senior Member AIAA

‡ Computer Scientist

§ Senior Research Scientist